## WHAT IS CLAIMED IS:

- 1. A lock-free implementation of a shared object:
- that is population-oblivious,
- for which consumption of storage is adaptive independent of any historical maximum, and
- for which failure of a thread does not prevent all future reclamation of the storage by a non-failed thread,
- wherein the adaptivity of storage consumption is achieved through explicit reclamation and without resort to garbage collection.
- 2. The lock-free shared object implementation of claim 1, employed in the implementation of a garbage collector.
- 3. The lock-free shared object implementation of claim 1, wherein concurrent operations on the shared object are mediated using only single-target synchronization primitives.
- 4. The lock-free shared object implementation of claim 3, wherein the single-target synchronization primitives include uses of a Compare-And-Swap (CAS) operation.
- The lock-free shared object implementation of claim 3,
  wherein the single-target synchronization primitives include uses of a Load-Linked (LL) and Store-Conditional (SC) operation pair.
- 6. The lock-free shared object implementation of claim 1, wherein the adaptivity includes adaptivity as a function of size of the shared object; and
- wherein the independence includes independence at least of a historical maximum of object size.
- 7. The lock-free shared object implementation of claim 1,

- wherein the adaptivity includes adaptivity as a function of a number of processes that concurrently access the shared object; and wherein the independence includes independence at least of a historical maximum for the number of processes that concurrently access the shared object.
- 8. The lock-free shared object implementation of claim 1, wherein time complexity of operations on the shared object is also adaptive.
- 9. The lock-free shared object implementation of claim 1, wherein the shared object includes nodes organized with predecessor and successor relations thereamongst, wherein in-degree of each one of the nodes is at most one (1) and an immediate predecessor one of the nodes can be identified from a successor one of the nodes.
- 10. The lock-free shared object implementation of claim 1, wherein the shared object includes nodes organized as a list.
- 11. The lock-free shared object implementation of claim 1, wherein the shared object includes nodes organized as a hierarchy.
- 12. A computer readable medium encoding of an implementation of a dynamically sizable shared object, the encoding comprising:
  - a definition of a node including a forward-direction pointer encoded integrally with a respective counter, the node instantiable as part of the shared object; and
  - a functional encoding of lock-free operations executable to traverse the shared object, each of the operations reading and atomically updating respective integrally encoded counters coincident with a related traversal, the functional encodings including both a forward-direction, counter-incrementing, pointer operation and a reverse-direction, counter-decrementing operation,

Attorney Docket No.: 004-8193

wherein corresponding executions of the forward-direction operation and the reverse-direction operation both atomically read and update the counter integrally encoded with the corresponding forward-direction pointer.

- 13. An encoding of a shared object implementation, as recited in claim 12, wherein the node definition further includes a reverse-direction pointer; and wherein the reverse-direction, counter-decrementing operation follows one of the reverse-direction pointers, but decrements the counter encoded with the corresponding forward-direction pointer.
- 14. An encoding of a shared object implementation, as recited in claim 13, wherein the related traversals include traversals of corresponding ones of the forward-direction and reverse direction pointers.
- 15. An encoding of a shared object implementation, as recited in claim 12, wherein the reverse direction operation uses node information recorded as part of the execution of the forward direction operation.
- 16. An encoding of a shared object implementation, as recited in claim 12, wherein the shared object implements a collect object.
  - 17. An encoding of a shared object implementation, as recited in claim 16, wherein the operations include a collect operation that employs forward-direction operations as it searches through nodes of the shared object.
  - 18. An encoding of a shared object implementation, as recited in claim 17, wherein the collect operation employs reverse-direction operations to remove nodes of the shared object.
  - 19. An encoding of a shared object implementation, as recited in claim 16, wherein the forward-direction operations include register operations; and wherein the reverse-direction operations include deregister operations.
  - 20. An encoding of a shared object implementation, as recited in claim 16,

Attorney Docket No.: 004-8193

wherein the reverse-direction operations include cleanup operations.

21. An encoding of a shared object implementation, as recited in claim 16, wherein the encoding of the counter distinguishes contributions of collect operations from those of non-collect operations.

- 22. An encoding of a shared object implementation, as recited in claim 12, wherein the shared object implements a space adaptive guard array for a value recycling solution.
- 23. An encoding of a shared object implementation, as recited in claim 12, wherein the shared object implements a space adaptive renaming solution.
- 24. An encoding of a shared object implementation, as recited in claim 12, wherein the atomic read and update functionality is provided using a single target synchronization primitive.
- 25. An encoding of a shared object implementation, as recited in claim 24, wherein the single-target synchronization primitive includes a Compare-And-Swap (CAS) operation.
- 26. An encoding of a shared object implementation, as recited in claim 24, wherein the single-target synchronization primitive includes a Load-Linked (LL) and Store-Conditional (SC) operation pair.
- 27. An encoding of a shared object implementation, as recited in claim 12, wherein the atomic read and update functionality is provided using an atomic operation and operations on the shared object are wait-free.
- 28. An encoding of a shared object implementation, as recited in claim 12, wherein, when instantiated as part of the shared object, the nodes are organized with predecessor and successor relations thereamongst, and

- wherein in-degree of each one of the nodes is at most one (1) and an immediate predecessor one of the nodes can be identified from a successor one of the nodes.
- 29. An encoding of a shared object implementation, as recited in claim 12, wherein, when instantiated as part of the shared object, the nodes are organized as a list.
- 30. An encoding of a shared object implementation, as recited in claim 12, wherein, when instantiated as part of the shared object, the nodes are organized as a hierarchy.
- 31. An encoding of a shared object implementation, as recited in claim 12, wherein the implementation is population oblivious and for which consumption of storage is adaptive independent of any historical maximum.
- 32. An encoding of a shared object implementation, as recited in claim 12, wherein failure of a thread does not prevent all future reclamation, by a non-failed thread, of storage associated with the shared object.
- 33. A method of implementing a population-oblivious, dynamically sizable, lock-free shared object, the method comprising:
  - defining of nodes of the shared object to include a forward-direction pointer encoded integrally with a respective counter;
  - defining operations executable to traverse the shared object, each of the operations reading and atomically updating respective integrally encoded counters coincident with a related traversal operation, the encodings including both a forward-direction, counter-incrementing, pointer operation and a reverse-direction, counter decrementing operation, wherein corresponding executions of the forward-direction operation and the reverse-direction operation both atomically read and update the counter integrally encoded with the corresponding forward-direction pointer.

- 34. The method of claim 33, further comprising:
- defining the nodes of the shared object to further include a reverse-direction pointer,
- wherein the reverse-direction, counter-decrementing operation follows one of the reverse-direction pointers, but decrements the counter encoded with the corresponding forward-direction pointer.
- 35. The method of claim 33, further comprising:
- recording node information as part of execution of the forward-direction operation; and
- using the recorded node information for traversal by the reverse-direction operation.
- 36. The method of claim 33,
- wherein the population-oblivious, dynamically sizable, lock-free shared object implements a collect object.
- 37. The method of claim 36, further comprising:
- distinguishing, in the encoding of the counter, contributions of collect operations from those of non-collect operations.
- 38. The method of claim 33,
- wherein the population-oblivious, dynamically sizable, lock-free shared object implements a space adaptive guard array for a value recycling solution.
- 39. The method of claim 33,
- wherein the atomic read and update functionality is provided using a single target synchronization primitive.
- 40. The method of claim 39, wherein individual instances of the single-target synchronization primitive include one of:
  - a Compare-And-Swap (CAS) operation; and
  - a Load-Linked (LL) and Store-Conditional (SC) operation pair.

# 41. The method of claim 33,

wherein, when instantiated as part of the shared object, the nodes are organized with predecessor and successor relations thereamongst, and wherein in-degree of each one of the nodes is at most one (1) and an immediate predecessor one of the nodes can be identified from a successor one of the nodes.

## 42. The method of claim 33,

wherein, when instantiated as part of the shared object, the nodes are organized as a list.

#### 43. The method of claim 33,

wherein, when instantiated as part of the shared object, the nodes are organized as a hierarchy.

# 44. The method of claim 33,

wherein the shared object is adaptive independent of any historical maximum.

## 45. The method of claim 33,

wherein failure of a thread that operates on the shared object does not prevent all future reclamation, by a non-failed thread, of storage associated with the shared object.

# 46. An apparatus comprising:

one or more processors for executing threads of a computation; shared storage accessible by the one or more processors; and means for instantiating in shared storage a lock-free, population-oblivious,

- shared object for which consumption of storage is adaptive independent of any historical maximum, and for which failure of any
- one of the threads does not prevent all future reclamation of the storage by a non-failed one of the threads.